一、DOM:

文档对象模型(Document Object Model，简称DOM)，是W3C组织推荐的处理可扩展标记语言(HTML或者XML)的标准编程接口。

作用：W3C已经定义了一系列的DOM接口，通过这些DOM接口可以改变网页的内容、结构和样式。

1.对于JavaScript，为了能够使JavaScript操作HTML，JavaScript就有了一套自己的dom编程接口。

2.对于HTML, dom使得html形成一棵dom树。包含文档、元素、节点

我们获取过来的DOM元素是一个对象(object)，所以称为文档对象模型

1、DOM操作

关于dom操作，我们主要针对于元素的操作。主要有创建、增、删、改、查、属性操作、事件操作。

1 创建htnl元素

1. document.write
2. innerHTML
3. createElement

区别:

1. document.write是直接将内容写入页面的内容流，但是文档流执行完毕，则它会导致页面全部重绘
2. innerHTML是将内容写入某个 DoM节点，不会导致页面全部重绘
3. innerHTML创建多个元素效率更高(不要拼接字符串，采取数组形式拼接），结构稍微复杂
4. createElement( )创建多个元素效率稍低一点点，但是结构更清晰

总结：不同浏览器下，innerHTML效率要比 creatElement高

2 增

1. appendChild
2. insertBefore

3 删

1. removeChild

4 改

主要修改DOM的元素属性，DOM元素的内容、属性,表单的值等

1. 修改元素属性: src、href、title等
2. 修改普通元素内容:innerHTML、 innerText
3. 修改表单元素: value、type、disabled等
4. 修改元素样式: style、className

5 查

主要获取查询DOM的元素

1. DOM提供的API方法: getElementByld、getElementsByTagName古老用法不太推荐
2. H5提供的新方法: querySelector、querySelectorAll提倡

3. 利用节点操作获取元素：父(parentNode)、子(children)、兄(previousElementSibling、nextElementSibling)提倡

6 属性操作

主要针对于自定义属性。

1. setAttribute :设置dom的属性值
2. getAttribute :得到dom的属性值
3. removeAttribute移除属性

7 事件操作

给元素注册事件，采取 事件源.事件类型 = 事件处理程序

鼠标事件	触发条件
onclick	鼠标点击左键触发
onmouseover	鼠标经过触发
onmouseout	鼠标离开触发
onfocus	获得鼠标焦点触发
onblur	失去鼠标焦点触发
onmousemove	鼠标移动触发
onmouseup	鼠标弹起触发
onmousedown	鼠标按下触发

8 注册事件（绑定事件）

1.1注册事件概述

给元素添加事件，称为注册事件或者绑定事件。注册事件有两种方式:传统方式和方法监听注册方式。

传统注册方式

利用on开头的事件onclick:

```
<button onclick= “alert(hi~")" ></button>
btn.onclick = function( ){}
```

特点：注册事件的唯一性

同一个元素同一个事件只能设置一个处理函数，最后注册的处理函数将会覆盖前面注册的处理函数

方法监听注册方式

- w3c标准推荐方式
- addEventListener()它是一个方法
- IE9之前的E不支持此方法，可使用attachEvent()代替
- 特点:同一个元素同一个事件可以注册多个监听器
- 按顺序进行

addEventListener事件监听方式

eventTarget.addEventuistener(type, listener [, usecapture])
eventTarget.addEventListener()方法将指定的监听器注册到eventTarget(目标对象）上，当该对象触发指定的事件时，就会执行事件处理函数。
该方法接收三个参数：

- type：事件类型字符串，比如click、mouseover ，注意这里不要带on
- listener：事件处理函数，事件发生时，会调用该监听函数
- useCapture：可选参数，是一个布尔值，默认是false。

二、BOM/DOM盒子

1、offset系列：

1.1 offset 概述

offset 翻译过来就是偏移量，我们使用 offset 系列相关属性可以动态的得到该元素的位置（偏移）、大小等。

- 获得元素距离带有定位父元素的位置
- 获得元素自身的大小（宽度高度）
- 注意：返回的数值都不带单位

offset 系列常用属性：

offset系列属性	作用
element.offsetParent	返回作为该元素带有定位的父级元素 如果父级都没有定位则返回body
element.offsetTop	返回元素相对带有定位父元素上方的偏移
element.offsetLeft	返回元素相对带有定位父元素左边框的偏移
element.offsetWidth	返回自身包括padding 、 边框、内容区的宽度，返回数值不带单位
element.offsetHeight	返回自身包括padding、边框、内容区的高度，返回数值不带单位

1.2 offset 与 style 区别

offset

- offset 可以得到任意样式表中的样式值
- offset 系列获得的数值是没有单位的
- offsetWidth 包含padding+border+width
- offsetWidth 等属性是只读属性，只能获取不能赋值
- 所以，我们想要获取元素大小位置，用offset更合适

style

- style 只能得到行内样式表中的样式值
- style.width 获得的是带有单位的字符串
- style.width 获得不包含padding和border 的值
- style.width 是可读写属性，可以获取也可以赋值
- 所以，我们想要给元素更改值，则需要用style改变

2. 元素可视区 client 系列

client 翻译过来就是客户端，我们使用 client 系列的相关属性来获取元素可视区的相关信息。通过 client 系列的相关属性可以动态的得到该元素的边框大小、元素大小等。

client系列属性	作用
element.clientTop	返回元素上边框的大小
element.clientLeft	返回元素左边框的大小
element.clientWidth	返回自身包括padding 、 内容区的宽度，不含边框，返回数值不带单位
element.clientHeight	返回自身包括padding 、 内容区的高度，不含边框，返回数值不带单位

2、盒模型

1. 盒模型信息获取：

节点.getBoundingClientRect()：得到一个元素的尺寸和坐标位置；

a) left和top对应的是左上角x,y坐标；

b) right和bottom对应的是右下角x,y坐标；

c) 如果遇到浏览器返回的ClientRect里面如果没有width和height， 可以通过上面的坐标关系计算出来；

2. 得到一个盒子的相关尺寸信息属性：

a) offsetWidth：本身宽度+边框线+左右内边距；

b) offsetHeight：本身高度+边框线+上下内边距；

c) offsetTop：相对有定位属性的父节点上偏移量；

d) offsetLeft：相对有定位属性的父节点左偏移量；

e) clientWidth：本身的宽度+左右内边距；

f) clientHeight：本身的高度+上下内边距；

g) clientTop：上边框线的宽度；

h) clientLeft：左边框线的宽度；

i) scrollWidth：盒子的实际宽度(包括不可见部分)；

j) scrollHeight：盒子的实际高度(包括不可见部分)；

k) scrollTop：滚动条向下滚动的距离；

l) scrollLeft：滚动条向右滚动的距离；

m) window.innerHeight：浏览器窗口可见区域高度；

n) window.innerWidth：浏览器窗口可见区域宽度;

3、让窗口滚动到指定位置

```
<div class="content"></div>
```

```
    <img src="https://img0.baidu.com/it/u=1480630091,1335611843&fm=253&fmt=auto&app=138&f=JPEG?w=588&h=441" alt=""
        srcset="">
    <div class="content"></div>

// 滚到指定坐标，前提是页面够长 要能滚动。
    <script>
        // 1. window.scroll()  此方法接收两个参数，依次为X坐标和Y坐标；设置滚动条的偏移位置

        // 2. window.scrollTo() 此方法和scroll()作用一样，都是设置滚动条的偏移位置。

        // 3. window.scrollBy() 此法发同样接收两个参数，不过参数分别为X轴的偏移量和Y轴的偏移量，并且可以增加或者减少。

        scroll(0, 200); // 设置滚动条Y轴位置在200像素的地方。比如：当前坐标为0，执行后便是200，当前坐标为100，执行后是200。

        scrollTo(0, 200); // 同scroll()方法。

        scrollBy(0, 200); // 使得滚动条Y轴的位置，在当前的基础上增加200。比如：当前Y轴位置为0，执行后便是200；当前为100，执行后便是300。
        // window.scroll(x, y); 让滚动条滚动到距离浏览器窗口目标的距离处
        // window.scrollBy(0,200);使得滚动条Y轴的位置，在当前的基础上增加200。比如：当前Y轴位置为0，执行后便是200；当前为100，执行后便是300。
        let img = document.querySelector('img');
        img.onclick = function () {
            // window.scroll(0, 200);
            // window.scrollBy(0, 200);
            window.scrollTo(0, 200);
        }
    </script>
// 滚动到指定元素可见，调用元素的底部会尽量与视口的顶部齐平
el.scrollIntoView(true)
// 默认为true
// 如果传递参数false，则将元素的下边缘和视口的下边缘对齐
```

页面滚动的距离

```
window.pageXOffset/window.pageYOffset // bom操作，IE8及IE8以下不兼容
document.body.scrollLeft/scrollTop
```

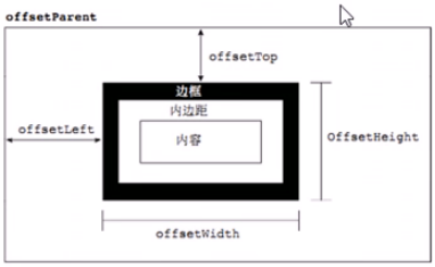
```
document.documentElement.scrollLeft/scrollTop

// 兼容做法: sY=window.pageYOffset || document.body.scrollTop || document.documentElement.scrollTop;
```

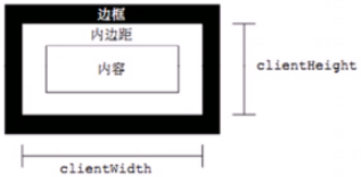
窗口的可视区域尺寸

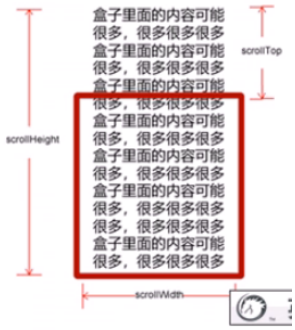
```
    <div class="demo"> </div>

    <script>
        let demo = document.querySelector('.demo');
        demo.scrollIntoView(true)
        console.log(window.pageXOffset);

        console.log(window.innerHeight);// 浏览器窗口的内部高度
        console.log(window.innerWidth); // 浏览器窗口的内部宽度

        // 兼容写法
        let w = window.innerWidth || document.documentElement.clientHeight || document.body.clientWidth;
        let h = window.innerHeight || document.documentElement.clientHeight || document.body.clientHeight;
        console.log(w, h);

        // 拓展
        /*
          Chrome/FF/Safari/opera
          对这些浏览器而言，window有个属性innerWidth/innerHeight包含的是
          整个文档的可视区域尺寸，注意，这个尺寸是包含滚动条大小的。
          document.documentElement属性引用了作为文档根元素的html标记，
          document.body属性引用了body标记
        */
    </script>
```

三大系列总结

三大系列大小对比	作用
element.offsetWidth	返回自身包括padding 、 边框、内容区的宽度，返回数值不带单位
element.clientWidth	返回自身包括padding 、 内容区的宽度，不含边框，返回数值不带单位
element.scrollWidth	返回自身实际的宽度，不含边框，返回数值不带单位







mouseenter 和mouseover的区别

mouseenter 鼠标事件

- 当鼠标移动到元素上时就会触发 mouseenter 事件
- 类似 mouseover，它们两者之间的差别是
- mouseover 鼠标经过自身盒子会触发，经过子盒子还会触发。 mouseenter 只会经过自身盒子触发
- 之所以这样，就是因为mouseenter不会冒泡

4. 动画函数封装

4.1 动画实现原理

核心原理：通过定时器 setInterval() 不断移动盒子位置。

实现步骤：

1. 获得盒子当前位置
2. 让盒子在当前位置加上1个移动距离
3. 利用定时器不断重复这个操作
4. 加一个结束定时器的条件
5. 注意此元素需要添加定位，才能使用element.style.left

```
    <style>
        div {
            position: absolute;
            left: 0;
            width: 100px;
            height: 100px;
            background-color: pink;
        }
    </style>
</head>

<body>
    <div>盒子</div>
    <script>
        //动画要设置定位
        //动画封装: obj目标对象，target目标位置
        function animate(obj, target) {
            let timer = setInterval(function () {
                if (obj.offsetLeft >= target) {
                    clearInterval(timer);
                }
                obj.style.left = obj.offsetLeft + 2 + 'px';
```

```
            }, 30)
        }
        let div = document.querySelector('div');
        //调用函数
        animate(div, 300);
    </script>
</body>
```

效率提升：给不同元素指定不同的定时器（解决元素的速度会越来越快，因为开启了太多的定时器的问题）

```
    <style>
        div {
            position: absolute;
            left: 0;
            width: 100px;
            height: 100px;
            background-color: pink;
        }

        span {
            display: block;
            width: 200px;
            height: 200px;
            position: absolute;
            left: 0;
            top: 150px;
            background-color: aquamarine;
        }
    </style>
</head>

<body>
    <button>点击</button>
    <div>盒子</div>
    <span>span</span>
    <script>
        //动画要设置定位

```

```
        //let obj = {};
        //obj.name = 'andy';

        //动画封装: obj目标对象，target目标位置
        //给不同元素指定不同的定时器
        function animate(obj, target) {
            //当我们不断的点击按钮，这个元素的速度会越来越快，因为开启了太多的定时器
            //解决方案就是让我们元素只有一个定时器执行
            //先清除以前的定时器，只保留当前的一个定时器执行
            clearInterval(obj.timer);
            obj.timer = setInterval(function () {
                if (obj.offsetLeft >= target) {
                    clearInterval(obj.timer);
                }
                obj.style.left = obj.offsetLeft + 2 + 'px';
            }, 30)
        }
        let div = document.querySelector('div');
        let span = document.querySelector('span');
        let btn = document.querySelector('button');
        //调用函数
        animate(div, 300);
        btn.addEventListener('click', function () {
            animate(span, 300);
        })
    </script>
```

缓动动画

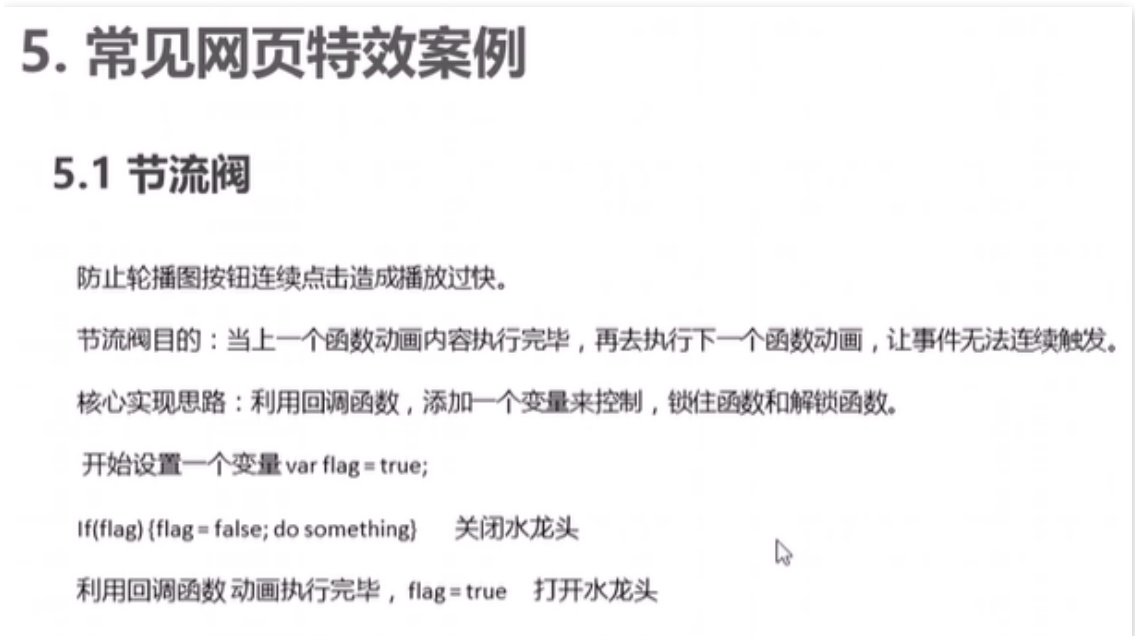
```
    <style>
        span {
            display: block;
            width: 150px;
            height: 150px;
            position: absolute;
            left: 0;
            top: 50px;
            background-color: aquamarine;
        }
```

```
    </style>
</head>

<body>
    <button class="btn500">点击500</button>
    <button class="btn800">点击800</button>
    <span>span</span>

    <script>
        // 缓动动画公式：（目标值-现在的位置）/ 10
        function animate(obj, target) {
            clearInterval(obj.timer);

            obj.timer = setInterval(function () {
                //步长值写到定时器里面
                //把步长值变为整数
                let step = (target - obj.offsetLeft) / 10;
                step = step > 0 ? Math.ceil(step) : Math.floor(step);

                if (obj.offsetLeft >= target) {
                    clearInterval(timer);
                }
                obj.style.left = obj.offsetLeft + step + 'px';
            }, 30)
        }

        let span = document.querySelector('span');
        let btn500 = document.querySelector('.btn500');
        let btn800 = document.querySelector('.btn800');

        btn500.addEventListener('click', function () {
            animate(span, 500);
        })
        btn800.addEventListener('click', function () {
            animate(span, 800);
        })
    </script>
</body>
```

添加回调函数：回调函数写到定时器结束里面

```
    <style>
        span {
            display: block;
            width: 150px;
            height: 150px;
            position: absolute;
            left: 0;
            top: 50px;
            background-color: aquamarine;
        }
    </style>
</head>

<body>
    <button class="btn500">点击500</button>
    <button class="btn800">点击800</button>
    <span>span</span>

    <script>
        function animate(obj, target, callback) {
            clearInterval(obj.timer);

            obj.timer = setInterval(function () {
                let step = (target - obj.offsetLeft) / 10;
                step = step > 0 ? Math.ceil(step) : Math.floor(step);
                if (obj.offsetLeft >= target) {
                    clearInterval(obj.timer);
                    //回调函数写到定时器结束里面
                    // if (callback) {
                    //     callback();
                    // }
                    callback && callback();

                }
                obj.style.left = obj.offsetLeft + step + 'px';
            }, 30)
        }

```

```
        let span = document.querySelector('span');
        let btn500 = document.querySelector('.btn500');
        let btn800 = document.querySelector('.btn800');

        btn500.addEventListener('click', function () {
            animate(span, 500, function () {
                span.style.backgroundColor = 'pink';
            });
        })
        btn800.addEventListener('click', function () {
            animate(span, 800, function () {
                span.style.backgroundColor = 'red';
            });
        })
    </script>
</body>
```



轮播图案例：JS代码

```
window.addEventListener('load', function () {
    //1获取元素
    let sj_l = document.querySelector('.sj-l');
    let sj_r = document.querySelector('.sj-r');
    let lb = document.querySelector('.lb-right');
    // 每次图片要滚动的宽度
```

```
    let lbulWidth = lb.offsetWidth;
    //2 鼠标经过就显示，隐藏
    lb.addEventListener('mousemove', function () {
        sj_l.style.display = 'block';
        sj_r.style.display = 'block';
        clearInterval(timer);
        timer = null;
    })
    lb.addEventListener('mouseleave', function () {
        sj_l.style.display = 'none';
        sj_r.style.display = 'none';
        timer = setInterval(function () {
            //手动调用点击事件
            sj_r.click();
        }, 2000);
    })
    //3 动态生成小圆圈有几张图片，我就生成几个小圆圈
    let ul = lb.querySelector('ul');
    let ol = lb.querySelector('.yq');
    for (let i = 0; i < ul.children.length; i++) {
        //创建li
        let li = document.createElement('li');
        //记录当前小圆圈的索引号，通过自定义属性来做
        li.setAttribute('index', i);
        //插入ol中
        ol.appendChild(li);
        //4 排他思想，生成li的同时直接绑定点击事件
        li.addEventListener('click', function () {
            for (let j = 0; j < ol.children.length; j++) {
                //排除其他li
                ol.children[j].className = '';
            }
            //留下自己
            this.className = 'current';
            //5 点击小圆圈，移动图片，移动的是ul
            //ul移动的距离=小圆圈索引x图片的宽度，是负值
            //当我们点击某个小li，就拿到当前li的索引号
            let index = this.getAttribute('index');
            //当我们点击某个li时，就把它的索引号给num
            num = index;
```

```
            //当我们点击某个li时，就把它的索引号给circle
            circle = index

            animate(ul, -index * lbulWidth);
        })
    }
    //把ol里第一个li设置类名为current
    ol.children[0].className = 'current';
    //6 克隆第一张图片（li）放到ul最后面，不会增加小圆圈
    let first = ul.children[0].cloneNode(true);
    ul.appendChild(first);
    //7 点击右侧按钮，图片滚动一张
    let num = 0;
    let circle = 0;//声明一个变量控制小圆圈的播放
    //节流阀
    let flag = true;
    sj_r.addEventListener('click', function () {
        if (flag) {
            flag = false;//关闭节流阀
            //如果走到了最后一张图片，此时，我们的ul 要快速复原 left改为0
            if (num == ul.children.length - 1) {
                ul.style.left = 0;
                num = 0;
            }
            num++;
            animate(ul, -num * lbulWidth, function () {
                flag = true;//打开节流阀
            });
            //8 点击右侧按钮，小圆圈跟随一起变化，可以在声明一个变量控制小圆圈的播放
            circle++;
            //如果circle==4 说明走到最后我们克隆的这张图片了，我们应该复原
            if (circle == ol.children.length) {
                circle = 0;
            }
            //调用函数
            circleChange();
        }
    });
    //9 左边按钮
    sj_l.addEventListener('click', function () {
```

```
        if (flag) {
            flag = false;
            if (num == 0) {
                num = ul.children.length - 1;
                ul.style.left = - num * lbulWidth + 'px';
            }
            num--;
            animate(ul, -num * lbulWidth, function () {
                flag = true;//打开节流阀
            });
            //左侧按钮，小圆圈跟随一起变化，可以在声明一个变量控制小圆圈的播放
            circle--;
            //(1)如果circle < 0 说明第一张图片，则小图片要改为第4个小圆圈（索引号为3）
            if (circle < 0) {
                circle = ol.children.length - 1;
            }
            //(2)三目
            //circle = circle < 0 ? ol.children.length - 1 : circle;

            //调用函数
            circleChange();
        }

    });
    //先清除其余小圆圈的current类名，封装成函数
    function circleChange() {
        for (let n = 0; n < ol.children.length; n++) {
            ol.children[n].className = '';
        }
        ol.children[circle].className = 'current';
    }
    //10 自动播放
    let timer = setInterval(function () {
        //手动调用点击事件
        sj_r.click();
    }, 2000);
});
```

1. 触屏事件

1.1 触屏事件概述

移动端浏览器兼容性较好，我们不需要考虑以前 JS 的兼容性问题，可以放心的使用原生 JS 书写效果，但是移动端也有自己独特的地方。比如触屏事件 touch（也称触摸事件），Android 和 IOS 都有。

touch 对象代表一个触摸点。触摸点可能是一根手指，也可能是一根触摸笔。触屏事件可响应用户手指（或触控笔）对屏幕或者触控板操作。

常见的触屏事件如下：

触屏touch事件	说明
touchstart	手指触摸到一个 DOM 元素时触发
touchmove	手指在一个 DOM 元素上滑动时触发
touchend	手指从一个 DOM 元素上移开时触发

1.2 触摸事件对象（TouchEvent）

TouchEvent 是一类描述手指在触摸平面（触摸屏、触摸板等）的状态变化的事件。这类事件用于描述一个或多个触点，使开发者可以检测触点的移动，触点的增加和减少，等等

touchstart、touchmove、touchend 三个事件都会各自有事件对象。

触摸事件对象重点我们看三个常见对象列表：

触摸列表	说明
touches	正在触摸屏幕的所有手指的一个列表
targetTouches	正在触摸当前 DOM 元素上的手指的一个列表
changedTouches	手指状态发生了改变的列表，从无到有，从有到无变化